C programnyelv alapjai Arduinohoz

Piarista Gimnázium, robotika szakkör

1. Arduino kód felépítése, általános szabályok

- A kód föntről lefelé fut le, a legfontosabb blokkok (függvények, ciklusok, stb.) kapcsos zárójelek közé íródnak.
- A két legfontosabb függvény a `void setup()` és a `void loop()`, ezek minden arduino kódban megtalálhatóak.
- Először a kontroller a `void setup()` függvény fut le, egyszer.
- Ezután a `void loop()` fut újra és újra, mindaddig, amíg az Arduino működik. Ha a programkód végére ér, elölről kezdi.

Az alábbi kód másodpercenként „köszön":

```
  void setup()
  {
      //az itt lévő kód egyszer fut le, az induláskor
      Serial.begin(9600);
  }

  void loop()
  {
      //az itt lévő kód újra és újra lefut, amíg valaki mást nem
mond
      Serial.println("hello");
      delay(1000);
  }
```

- **Fontos!** A C programnyelvben a parancsot tartalmazó sorok végére pontosvesszőt (;) teszünk.
- Különbség van a kis- és nagybetűk között, így pl. Janos és janos két különböző dolog.
- Ne használjunk ékezetes betűket.
- Megjegyzéseket a sorok végére `//Megjegyzés`, vagy több soron át `/* Megjegyzés */` formában tehetünk.

---

2. Változók

- Változókat használunk a program lefutása során használt értékek, szövegek, stb. tárolására és a velük való számolásra.
- Típusuk alapján sok féle változót elkülönítünk, itt csak a legegyszerűbbek vannak felsorolva.

a) Karakterváltozó (`char`)

- Egy nyolc biten tárolt szám, tehát 255 féle értéket vehet fel.
- Minden értékéhez tartozik egy egy karakter/írásjel is, az ASCII-tábla alapján. *(Ha érdekel, Google a barátod...)*
- Normál esetben elójeles, de megadható `unsigned char` alakban, ekkor csak pozitív értéket vehet fel.

b) Egész szám (`int`)

- Az Arduinon 16 bites szám, így értéke -32.768 és 32.767 között változhat, de törtrészt nem tartalmazhat.
- Ennek is van előjel nélküli csak pozitív alakja, az `unsigned int`
- Ha nagyobb értékre van szükségünk, `long` típusú változót kell használnunk, ez 32 biten van tárolva, így értéke -2.147.483.648 és 2.147.483.647 között mozoghat.
- A `long` előjel nélküli változata az `unsigned long`.

c) Szöveg (`String`)

- Ha nem csak egy karaktert, hanem azok egy sorát kell tárolni, `String` változó használunk.
- **Vigyázz!** Nagy S.
- Értékét annak megadásakor "idézőjelbe" tesszük.

d) Általános tudnivalók

Itt egy példa a fent leírt változókra:

```
int i; //Megadhatok egy változót érték nélkül, későbbi
használatra
int j = 20; //Vagy adhatok neki rögtön értéket

void setup()
{
    Serial.begin(9600);
    i = 5; //Adok egy értéket a változónak
}

void loop()
{
   String nev = "Elek"; //Függvényen belül is megadható változó
   Serial.println(nev); //A soros monitorra kiírja a nev
változóban tárolt értéket
   int k = i + j; //korábbi változók értékei is felhasználhatóak
   Serial.println(k); //A soros monitorra kiírja i+j=k értékét,
azaz 25-öt
}
```

- A fenti példában `i` és `j` globális változók, a függvényeken kívül vannak megadva, így minden függvény által elérhetőek.
- A `nev` egy lokális változó, a függvényen belül lett megadva, így csak ott elérhető. *(Így például a `void setup()`-ban nem használható.)*
- **Fontos!** Mielőtt műveleteket végeznénk egy változóval, értéket kell neki adni:
  - Ezt lehet rogön a változó megadásakor: `int i = 1;` (*deklarálás*)
  - vagy később: `int i;` (*definiálás*) majd később `i = 1;`
  - **Fontos!** Változónak értéket adni 1 db egyenlőségjellel lehet, a következő alakban: `valtozo = ertek;`
- Bizonyos neveket nem adhatunk a változóknak, például nincs `int int;`
- A lokális változók az őket tartalmazó fügvény lefutása után elvesztik értéküket.
- Egyszerre több változó is megadható, vesszővel elválasztva: `int a, b = 6;`
- **Fontos!** Ha csak egyszer kell egy számot használnom, nem kell rá változót bevezetnem, az is helyes, hogy `k = 20 + 5;`

---

3. Függvények

- Egy-egy sokszor elvégzendő műveletsort egyszerűsítenek le.
- Például, ha sokszor kellene egy adott műveletsort elvégeznem két számon és kiírnom a végeredményt, nem érdemes ezt mindenüvé beírni a kódban, hiszen később sokkal nehezebb kijavítani egy esetleges hibát 10 helyen... Ehelyett készítek egy függvényt, amit csak meg kell hívnom, hogy elvégezze a műveletsort.

Függvény deklarálása:

```
void szia(void)
{
    Serial.println("hello");
}
```

- Minden függvénynek van egy neve, egy visszatérítési értéke és bemeneti változói.
  - Amikor megadunk egy függvényt, először a visszatérített érték típusát adjuk meg: ez lehet `int`, `String`, vagy ha nem akarunk semmilyen értéket visszatéríteni, `void`.
  - Ezután megadjuk a függvény nevét, ez a fenti példában `szia`.
  - Végül zárójelben megadjuk a bemeneti változókat `(típus név)` alakban: ezek olyan változók, amelyeket a függvény meghívásakor adunk meg.
  - **Vigyázz!** Ennek a sornak a végén nincs pontosvessző!

Az alábbi függvény összead két számot, majd visszatéríti az összeget:

```
int osszeg(int a, int b) //oszeg nevű, egész számot visszatérítő
és két egész számot bekérő függvény
{
    int sum = a + b; //sum nevű változó egyenlővé tevése a két
bemeneti változóval
    return sum; //sum értékének visszatérítése
}
```

- Ha több bemeneti változót szeretnénk, vesszővel válasszuk el őket.
- Egy érték visszatérítéséhez a `return érték;` parancsot kell használni.

Függvény meghívása:

```
void setup()
{
Serial.begin(9600);
}

void loop()
{
    int i = osszeg(5, 6); //Bevezetek egy változót és egyenlővé
teszem 5 és 6 összegével
    Serial.println(i); //i értékének kiírása a soros monitorra
    delay(1000);
}

int osszeg(int a, int b) //A korábbi függvény
{
    int sum = a + b;
    return sum;
}
```

- A példában is látszik, hogy a függvény meghívása `függvény neve(bemeneti változó 1, bemeneti változó 2, stb);` alakban történik.
- **Fontos!** A bemeneti változók megfelelő típusúak legyenek!

---

4. Operátorok

a) Matematikai műveletek

- Összeadás, kivonás, szorzás, osztás: például `a + b`, `4 - 2`, `5 * x` és `i / 6`.
- Egy egy változóval akarunk elvégezni egy műveletet, pl: `a = a + 2;`, használható az `a += 2` kivejezés is. (Ugyanígy létezik `-=`, `*=` és `/=` operátor.)
- Praktikus rövidítés még az `a++;` , illetve az `a--` - ezek 1-gyel növelik, illetve csökkentik a változó értékét.

b) számok értékének összehasonlítása

- Ha azt kell megvizsgálni, két szám egyenlő-e, az `a == b;` operátort használjuk.
- Ha egynlőtlenséget vizsgáluk, az `a != b` operátort használjuk.
- Egyenlőtlenségek esetén a `a &lt; b`, `x &lt;= y`, `k &gt; 100` és `var &gt;= 5` alakú operátorokat használjuk.

c) if - else if - else

- Ha egy állításról el kell dönteni, hogy igaz-e az `if` (magyarul: „ha") állítást használjuk
- Az if szócska után zárójelbe írjuk a vizsgálandó állítást: `if(állítás) { ... }` majd a függvényekhez hasonlóan kapcsos zárójelek közé írjuk a kódot, ami akkor fut le, ha az állítás igaz.
- **Vigyázz!** Az if utáni sorban nincs pontosvessző!

Példa egy if állításra:

```
void setup()
{
    int ser = 1; //ser változó értéke legyen 1
    if (ser == 1) //Ha ser értéke 1, fusson le a kapcsos
zárójelek közötti kód
    {
        Serial.begin(9600);
    }
}
```

- A fenti program elindítja a soros kommunikációt, ha ser változó értéke egyenlő 1-gyel.

- Egy `if` állítás lezárása után opcionálisan elelyezhetünk tetszőleges számú `if else` állítást is, hasonló alakban.

- Ezeket fentről lefelé haladva értékeli ki a program, és amelyik igaz, azt lefuttatja.
- Az `if` és `if else` állítások után elhelyezhető egy `else` állítás is, ami akkor fut le, ha az előtte lévők közül egyik sem volt igaz.

Az alábbi kód x értékétől függően végez el feladatokat:

```
if(x == 0)
{
    //Az ide írt kód akkor fut le, ha x értéke 0
}
else if(x == 1)
{
    //Ez a kód pedig akkor fut le, ha x értéke 1
}
else
{
    //Ha x sem 0, sem 1, ez a kód fog lefutni
}
```

---

5. Ciklusok

- Ha valamilyen műveletsort egymás után többször el kell végeznünk, ciklusokat használunk.

a) `for` ciklus

- Eleinte vesz egy változót adott értéken, megnézi, megfelel-e egy megadott feltételnek, majd lefuttatja a ciklusban található kódot és megváltoztatja az eredeti változót. Ezt újra és újra megteszi, amígy a megadott feltétel teljesül.
- Használata: `for(változó = érték; feltétel; mit csináljon a változóval) {}`

Leggyakoribb alkalmazása:

```
void setup()
{
    int i;
    for(i = 1; i &amp;lt; 10; i++)
    {
        Serial.println("i kisebb, mint 10");
    }
}
```

- A fenti kód 9-szer írja ki a soros monitorra a megadott szövegét. Kezdetben vesz egy i változót, melynek az 1-es értéket adja, majd megnézi, i kisebb-e, mint 10. Ha igen, lefuttatja a kapcsos zárójelekbe írt kódot, megnöveli i-t 1-gyel, majd kezdi elölről.
- **Figyelem!** A `for` utáni zárójelen belül pontosvessző vannak, de a sor végén nincs semmi.

b) `while` ciklus

- Hasonló a `for` ciklushoz, akkor használjuk, ha nem megadott számú alkalommal kell lefuttatni egy kódot, hanem addig, amíg valamilyen feltétel teljesül.
- A `while` utáni zárójelben csak egy feltétel szerepel: `while(feltétel)` .

Például ez a kód addig újra és újra lefut, amíg a millis() *függvény visszatérítési értéke kisebb 3000-nél:*

```
void setup()
{
    int ido = millis();
    while(ido &amp;lt; 3000)
    {
        ido = millis();
    }
}
```

---

6. Tömbök

- Akkor használunk tömböket, ha egy bizonyos típusú változóból többet akarunk tárolni randezett alakban.
- Egy tömb olyan, mint egy polc, melyen számozott helyek vannak és minden helyre elhelyezhető egy változó.
- Tömböket a változókhoz hasonlóan adhatunk meg, ugyanúgy lehetnek globálisak és lokálisak.
- Először a tömbben tárolt változók típusát, majd a tömb nevét és elemeinek számát szögletes zárójelben, végül pedig magukat az elemeket adjuk meg: `int szamok[5] = {2, 4, 3, 0, 4};`
- Ha egy tömb valamely elemét el akarjuk érni, megadjuk a tömb nevét, illetve az elérni kívánt elem indexét.
- **Figyelem!** A tömb látszólag első elemének indexe 0!

Néhány példa tömbök elemeinek elérésére:

```
int tomb[3] = {12, 34, 100}; //Egy három int változót tartalmazó tömb létrehozása
int elso = tomb[0]; //Egy elso nevű egész szám egyenlővé tétele a tömb első, azaz nulladik elemével
int masodik = tomb[1]; //Ugyanez a második, azaz első elemével
Serial.println(tomb[3]); //A harmadik elemet pedig kiírjuk a soros monitorra
```

---

7. Two Player Reactor játékhoz szükséges kiegészítések

a) Soros monitor

- Az USB összeköttetésen át az Arduino képes egy, a számítógépre telepített speciális programmal kommunikálni: ez a soros monitor.
- A kommunikáció megkezdéséhez a `Serial.begin(9600);` parancs futtatására van szükség.
- A `Serial.println(valami);` parancs a zárójelbe tett kifejezést, vagy annak értékét írja ki a számítógépre, új sorban. *(Azonos sorba a `Serial.print();` paranccsal lehet írni)*

b) 1602 LCD

- Az Arduino projektek körében leginkább elterjedt folyadékkristályos kijelző, 16x2 karaktert képes megjeleníteni, kezeléséhez könyvtárak használatára van szükség.
- Létezik hozzá I2C protokollt használó vezérlő, melynek használatával kevesebb digitális portot foglal el.
- A kód lefutásának elején a kijelzőt inicializálni kell, ez a projekt szempontjából nem fontos.
- A kijelzőt kiüríteni az `lcd.clear();` paranccsal lehet.
- Mielőtt írunk a kijelzőre, be kell állítani a kurzor helyét az `lcd.setCursor(oszlop, sor);` paranccsal.
- **Figyelem!** Az első sor a nulladik, a második az első. Ugyanez a helyzet az oszlopokkal.
- A kijelzőre írni az `lcd.print(valami);` paranccsal lehet.

Egy példa az LCD használatára:

```
void loop()
{
    int x = 5;
    lcd.clear(); //Mielőtt új dolgokat írnánk rá, ki kell ürítsük
a kijelzőt
    lcd.setCursor(0, 0);
    lcd.print("x értéke:"); //Az első sor elejére egy szöveget
írunk
    lcd.setCursor(0, 1);
    lcd.print(x); //A második sor végére pedig kiírjuk x értékét
}
```

c) Ki- és bemenetek kezelése

- A kód elején, a `void setup()`-ban meg kell adnunk, hogy melyik pint milyen módon akarjuk használni.
- Ha kimenetként használnánk, a `pinMode(pin száma, OUTPUT)` parancsot használjuk.
- Ha bemenetként használnánk, például egy gomb figyelésére, `pinMode(pin száma, INPUT)`-nak kell beállítani.
- A kimenetnek beállított digitális pinek értékét a `digitalWrite(pin száma, érték)` paranccsal állíthatjuk be, ahol az érték `HIGH`, vagy `LOW` lehet.
- A bemenetek beolvasása a `digitalRead(pin száma)` paranccsal történik.

A fent leírtak gyakorlati alkalmazása:

```
void setup()
{
    pinMode(2, OUTPUT); //A 2-es pin legyen kimenet
    pinMode(3, INPUT); //A 3-as pin legyen bemenet
}

void loop()
{
    if(digitalRead(3) == HIGH) //Ha a 3-as pinről beolvasott
érték magas, azaz a pin áram alatt van...
    {
        digitalWrite(2, HIGH); //Legyen a 2-es pin értéke is
magas!
    }
    else
    {
        digitalWrite(2, LOW); //Egyéb esetben legyen alacsony
    }
}
```

d) Random számok

- A számítógépek nem tudnak a hasukra ütni és véletlenszerűen mondani egy számot, szükségük van egy kiindulási értékre, amiből ezt képzik.
- Az Arduinonál ezt a `randomSeed(szám);` sorral adhatjuk meg: ez a szám legyen egy `analogRead(0);` paranccsal a 0-s analóg pinről beolvasott elektromos zaj.
- Ebből már lehet random számot generálni, méghozzá a `random(x, y);` paranccsal.
- **Vigyázz!** Az első értéket felveheti a szám, a másodikat nem, például `random(1,5)` értéke lehet 1, 2, 3, vagy 4, de nem 5!

e) Időmérés

- Az Arduino méri az elindulása óta eltelt időt, amit a `millis()` függvénnyel kaphatunk meg, ezredmásodpercben mérve.
- **Figyelem!** A `millis()` függvény visszatérítési értékének típusa `unsigned long`, hiszen az `int` változó lehetséges értékén 32 másodperc után túlcsordulna.